Backtracking

From Wikipedia, the free encyclopedia
(Redirected from Backtracking search)

Backtracking is a strategy for finding solutions to constraint satisfaction problems. The term "backtrack" was coined by American mathematician D. H. Lehmer in 1950s.

Contents

Explanation

Constraint satisfaction problems are problems with a complete solution, where the order of elements does not matter. The problems consist of a set of variables each of which must be assigned a value, subject to the particular constraints of the problem. Backtracking attempts to try all the combinations in order to obtain a solution. Its strength is that many implementations avoid trying many partial combinations, thus speeding up the running-time.

Backtracking is closely related to combinatorial search.

Implementation

Backtracking algorithms try each possibility until they find the right one. It is a depth-first search of the set of possible solutions. During the search, if an alternative doesn't work, the search backtracks to the choice point, the place which presented different alternatives, and tries the next alternative. When the alternatives are exhausted, the search returns to the previous choice point and try the next alternative there. If there are no more choice points, the search fails.

This is usually achieved in a recursive function where each instance takes one more variable and alternatively assigns all the available values to it, keeping the one that is consistent with subsequent recursive calls. Backtracking is similar to a depth-first search but uses even less space, keeping just one current solution state and updating it.

In order to speed up the search, when a value is selected, before making the recursive call, the algorithm either deletes that value from conflicting unassigned domains (forward checking) or checks all the constraints to see what other values this newly-assigned value excludes (constraint propagation). This is the most efficient technique for certain problems like 0/1 knapsack and n-queen problem. It gives better results than dynamic programming for these problems.

Heuristics

Several heuristics are common to speed up the process. Because the variables can be processed in any order, it is generally efficient to try the most constrained ones first (i.e. the ones with fewest value options) as this prunes the search tree early (maximizes the impact of the current *early* choice).

When choosing which value to assign, many implementations use forward checking to see which value restricts the least number of values, in the anticipation that such a choice is a) more likely to preserve a possible solution and b) a solution has been found when the number of outstanding constraints has been reduced to zero.

Sophisticated backtracking implementations often use a bounding function, which checks whether it is possible to obtain a solution, for the current partial solution. Thus, a bounding test that detects partial solutions that fail can improve search efficiency. Because it is run often, possibly at every step, the computational cost of bounding needs to be minimal, otherwise the overall efficiency of the algorithm is not improved. Effective bounding functions are created in a similar way to other heuristic functions - by relaxing the rules of the problem slightly.

When backtracking is used in a constraint programming language, an added overhead occurs since information about the constraints, used by the constraint solver itself, needs to be updated as well. In these languages, a simple depth-first search is an adequate implementation technique, as used in Planner and Prolog.

In addition to retaining minimal recovery values used in backing up, backtracking implementations commonly keep a variable trail, to record value change history. An efficient implementation will avoid creating a variable trail entry between two successive changes when there is no choice point, as the backtracking will erase all of the changes as a single operation.

An alternative to the variable trail is to keep a time stamp of when the last change was made to the variable. The time stamp is compared to the time stamp of a choice point. If the choice point has an associated time later than that of the variable, it is unnecessary to revert the variable when the choice point is backtracked, as it was changed before the choice point occurred.

Applications

The most widespread use of backtracking is in the execution of regular expressions. For example, the simple pattern "a*a" will fail to match the sequence "a" without backtracking (because, on the first pass the "a" is eaten by the "a*" leaving nothing behind for the remaining "a" to match.)

Backtracking is used in the implementation of programming languages (such as Planner and Prolog) and other areas such as text parsing.

See also

- Ariadne's thread (logic)

Retrieved from "http://en.wikipedia.org/wiki/Backtracking"

Categories: Articles lacking sources from October 2006 | All articles lacking sources | Operations research | Search algorithms